# Le paquetage `makegobbler` v0.3 (2025-11-29)

Vincent Belaïche

12 décembre 2025

## Table des matières



## Liste des exemples

# 1 Introduction

Le but du paquetage `makegobbler` et de faciliter l'omission conditionnelle d'une portion de document. Un exemple valant mieux qu'un long discours :

```
\newif\ifgarde
...
\MGBkeep*\ifgarde
\LaMacroQuiGobeTout
Bla Bla bla, texte inséré conditionnellement seulement si
\gardetrue.
\LaMacroQuiGobeTout
```

EXEMPLE 1 – *Exemple introductif*

Dans l'exemple ci-dessus la macro `\LaMacroQuiGobeTout` est ci-après désigné par *délimiteur*, et on peut utiliser n'importe quel nom ⟨*délimiteur*⟩ de macro \⟨*délimiteur*⟩ aux lieux de `\LaMacroQuiGobeTout`. Dans le cas où `\conditionfalse` tout ce qui est entre les deux \⟨*délimiteur*⟩s est supprimé (gobé), et dans le cas inverse, quand `\conditiontrue`, c'est conservé (dégobé).
Dans le premier cas, le premier \⟨*délimiteur*⟩ est défini comme une macro de gobage, ou gobeur, gobant tout jusqu'à une marque de fin qui est le second \⟨*délimiteur*⟩ vu comme une string[1], et c'est la raison du nom `makegobbler` du paquetage, parce qu'il fait du premier \⟨*délimiteur*⟩ un gobeur.

# 2 Usage et limitations

## 2.1 Motivations

Le paquetage `makegobbler` n'est pas fait pour remplacer les constructions conditionnelles de TEX du genre de :

```
\newcommand*\Un{2}%
...
\ifnum\Un=1 C'est un\else C'est deux\fi
```

EXEMPLE 2 – *Gobage conditionnel classique à la TEX*

ni plus la macro `\ifthenelse` du paquetage LATEX `ifthen` qu'on peut utiliser ainsi :

```
\newcommand*\Un{2}%
...
\ifthenelse{\Un=1}{C'est un}{C'est deux}
```

EXEMPLE 3 – *Gobage conditionnel classique à la LATEX*

1. `string` en LATEX signifie une chaîne d'unités lexicales qui sont des caractères de catcode `other` ou `space`, dans le cas de `makegobbler` le \⟨*délimiteur*⟩ de fin ne comprend que des caractères de catcode `other`.

bien au contraire, `makegobbler` s'appuie justement sur ces constructions pour tester les conditions de gobage ou dégobage, et son objectif est plutôt de pallier certaines limitations du gobage effectué par ces méthodes classiques. L'exemple 4 illustre[2] cela : si vous décommentez le `\alternativetrue` il produit une erreur `You can't use 'macro parameter character #' in horizontal mode.`.

```
\newif\ifalternative
%\alternativetrue
...
Viens petite fille dans mon comic strip
...
\ifthenelse{\boolean{alternative}}{\verb+#!+}{SHEBAM !} POW !
BLOP ! WIZZ !
```

Exemple 4 – *Limitation d'*`\ifthenelse` *avec des caractères spéciaux*

Cela se produit parce que `\verb` fonctionne en changement le régime de catcode de ce qui suit, mais pour que cela fonctionne il faut que `\verb` soit exécuté[3]. Or si on passe `\verb+#!+` en argument à `\ifthenelse` le catcode de `#` est fixé au moment de la capture de l'argument, car celle-ci impose de passer le premier étage[4] de TEX, celui qui convertit les caractères en unités lexicales, et donc `#` conserve son catcode initial 6 (qui veut dire argument de macro) et `\verb` n'y pourra rien changer.

Le paquetage `makegobbler` résout le problème de l'exemple 4 parce que les sections dégobées ne sont pas passées en argument de macro, et d'ailleurs pour la même raison l'utilisation d'une condition à la TEX aurait aussi résolu le problème, comme dans l'exemple 5 ci-dessous :

```
\newif\ifalternative
\alternativetrue
...
Viens petite fille dans mon comic strip
...
\ifalternative\verb+#!+\else SHEBAM !\fi\space POW ! BLOP ! WIZZ !
```

Exemple 5 – *Caractères spéciaux avec une condition à la TEX*

Bien sûr dans ces exemples simplistes il aurait suffit d'écrire `\texttt{\#!}` au lieu de `\verb+#!+` et cela aurait aussi fonctionné, mais le prix de cela est d'écrire un code moins lisible.

Utiliser des conditions à la TEX n'est toutefois pas une solution ultime, en effet il reste des cas qui ne fonctionnent pas même avec les conditions à la TEX, l'exemple 6 en illustre un :

2. Aux non-francophones, c'est extrait des paroles de la chanson *Comic strip* de Serge Gainsbourg, la drôlerie des paroles vient qu'en franglais *strip* n'est connu que dans son sens *dévêtir* et non dans son sens *bande*.
3. Le deuxième étage de TEX, c-à-d. la gueule du lion.
4. les yeux du lion.

```
\newif\ifok
\oktrue
...
\ifok À minuit les carrosses redeviennent des
c\verb+\iftrue+illes.\fi
```

EXEMPLE 6 – *Limitations d'*`\if...`

Si vous mettez en commentaire le `\oktrue` alors il y aura une erreur de compilation `Incomplete \iffalse; all text was ignored after line ...` La raison pour laquelle cela échoue est que le `\ifok` pour gober tout ce qui suit jusqu'au `\fi` suivant teste tout de même s'il y a des unités lexicales de type `\if...`, de sorte à tomber sur le bon délimiteur de fin `\fi`. En faisant cela il considère par erreur que le `\iftrue` qui est passé dans `\verb+...+` attend un `\fi`, et donc il considère que le `\fi` suivant correspond à ce `\iftrue` et non à `\ifok`.

Avec `makegobbler` ce problème est résolu parce que l'utilisateur donne un délimiteur \⟨*délimiteur*⟩ arbitraire, et donc en cas de gobage avec imbrications de plusieurs sections en gobage/dégobage conditionnels il suffit d'utiliser un \⟨*délimiteur*⟩ propre à chaque niveau d'imbrication.

Dans cette rubrique j'ai surtout mentionné les problèmes qu'on peut avoir avec `\verb` que vous n'utilisez peut-être jamais. Sachez que c'est loin d'être le seul cas, et que toutes macros ou environnements qui jouent avec le régime de catcode sont sujets aux mêmes types de limitations, et d'ailleurs les questions posées par les débutants sur les forums ne concernent pas rarement la survenue de cette famille de problèmes. Notons pas exemple[5] que :

- `\includegraphics` avec les paquetages `graphicx,grffile` est capable de traiter correctement certains caractères spéciaux, comme par ex. `~`, au sein d'un nom de fichier en changeant son régime de catcode,
- `\href` et `\url` du paquetage `hyperref` sont également concernés pour traiter correctement certains caractères spéciaux au sein d'URLs, ou encore
- l'environnement `lstlisting` et la macro `\lstinline` de l'excellent paquetage `listings`, par exemple avec un listing en langage C on peut avoir typiquement des directives préprocesseur `#if CONDITION` qui poseraient problème si vous essayez de les dégober avec `\ifthenelse`.

## 2.2 Limitations de `makegobbler`

Si vous avez lu attentivement la rubrique §2.1 alors vous savez déjà qu'utiliser les macros de `makegobbler` au sein d'un argument de macro est voué à l'échec : comme `\verb` les macros de `makegobbler` jouent avec les catcodes, et donc produisent le même type de limitations.

Au sein de la plupart des environnements, cela au contraire va fonctionner. Et d'ailleurs si cela ne fonctionnait pas au sein de l'environnement `document`, alors le paquetage serait inutilisable. Toutefois avec certains environnements cela ne fonctionnera pas. En effet un environnement ⟨*env*⟩ peut-être de deux types :

- soit c'est un environnement *fil-de-l'eau*, dans ce cas le `\begin{`⟨*env*⟩`}` et sa contrepartie `\end{`⟨*env*⟩`}` sont juste comme deux macros qui activent et

5. Cette liste n'est en aucun cas exhaustive.

désactivent certaines conditions de traitement de ce qu'il y a entre les deux, par exemple la police, ou bien le régime de catcode ou encore le paramétrage de formation de paragraphes,
- soit c'est un environnement *aspirant*, cela signifie que tout ce qui se trouve entre `\begin{`⟨*env*⟩`}` et `\end{`⟨*env*⟩`}` est préalablement aspiré comme si c'était l'argument d'une macro de sorte à avoir une liste d'unités lexicales, et qu'ensuite cette liste est post-traitée.

Les macros de `makegobbler` ne vont donc fonctionner correctement pour gober/dégober des sections de code entre `\begin{`⟨*env*⟩`}` et `\end{`⟨*env*⟩`}` que si ⟨*env*⟩ est un environnement fil-de-l-eau, mais elles ne fonctionneront pas avec un environnement aspirant, puisqu'on se retrouve dans la même situation que si le code était au sein d'un argument de macro. Voici quelques environnements aspirants :

- la plupart des environnements du paquetage `amsmath`,
- tous les environnements créés avec la macro `\NewEnviron` du paquetage `environ`, et
- l'environnement `frame` de la classe `beamer`, sauf dans le cas où il reçoit l'argument optionnel `fragile`.

# 3 Options du paquetage

Le paquetage a deux options :

**ifthen** Juste une commodité pour charger le paquetage `ifthen`, ceci est nécessaire pour utiliser les macros de (dé)gobage conditionnel avec une condition à la LaTeX.

**tense** Ne pas utiliser sauf si vous savez ce que vous faites. Voir la documentation du code.

# 4 Interface utilisateur

## 4.1 Macros pour un gobage ou dégobage conditionnel

Le paquetage offre quatre macros pour le (dé)gobage conditionnel d'un bout de code LaTeX : `\MGBkeep`, `\MGBdrop`, `\MGBkeepelse`, et `\MGBdropelse`. Les deux premières utilisent deux `\`⟨*délimiteur*⟩s, et les deux dernières trois.

Chacune de ses macros a pour premier argument une condition. La syntaxe est la suivante :

```
\MGBkeep⟨étoile⟩⟨condition⟩\⟨délimiteur⟩
  ⟨dégobé⟩
\⟨délimiteur⟩
```

⟨*dégobé*⟩ peut être n'importe quoi entre les deux `\`⟨*délmiteur*⟩s, et il est dégobé si ⟨*condition*⟩ est vraie, et gobé sinon. Avec `\MGBdrop` c'est le contraire, la condition de débogage est [6] ¬⟨*condition*⟩.

Avec les deux autres macros, `\MGBkeepelse` et `\MGBdropelse`, on a deux branches, l'une contenant ⟨*gobé*⟩ et l'autre ⟨*dégobé*⟩ :

6. «¬$C$» signifie «non $C$».

```
\MGBkeepelse⟨étoile⟩⟨condition⟩\⟨délimiteur⟩
  ⟨dégobé⟩
\⟨délimiteur⟩
  ⟨gobé⟩
\⟨délimiteur⟩
```

Dans ce cas si ⟨*condition*⟩ est vraie, ⟨*dégobé*⟩ est dégobé, et ⟨*gobé*⟩ est gobé, et sinon, si ⟨*condition*⟩ est faux, c'est le contraire. En utilisant `\MGBdropelse`, la branche ⟨*gobé*⟩ correspondant au gobage sous ⟨*condition*⟩ est donnée en premier, et celle ⟨*dégobé*⟩ correspondant au dégobage en second.

```
\MGBdropelse⟨étoile⟩⟨condition⟩\⟨délimiteur⟩
  ⟨gobé⟩
\⟨délimiteur⟩
  ⟨dégobé⟩
\⟨délimiteur⟩
```

⟨*étoile*⟩ peut être soit `*` soit vide, dans le premier cas on dit que la macro est étoilée, et dans le second qu'elle est non-étoilée. Si la macro est non-étoilée, alors la condition est une condition à la LaTeX, c'est à dire une condition telle que la prend en premier argument la macro `\ifthenelse` du paquetage `ifthen`, par exemple[7] `{\boolean{true}}` ou `{\equal{`⟨*chaîne 1*⟩`}{`⟨*chaîne 2*⟩`}}`. Se reporter à la documentation de ce paquetage pour plus ample information. Avec les formes non étoilées il faut donc que le paquetage `ifthen` ait été chargé, ce que vous pouvez faire en passant l'option `ifthen` à `makeglobber`.

Avec la forme étoilée, le premier argument doit être une condition à la TeX, c'est à dire :

- soit une condition[8] `\iftoto` qui est instanciée avec une commande `\newif\iftoto` et qui peut être commutée avec les macros `\totofalse` et `\tototrue`,
- soit une condition qui utilise l'un des opérateurs de branchement conditionnel prédéfini dans TeX, par ex. `\ifmmode`, `{\ifnum`⟨*a*⟩=⟨*b*⟩␣`}`, `{\ifx\`⟨*a*⟩`\`⟨*b*⟩`}`, etc. Notez que dans les cas où ⟨*condition*⟩ consiste en plusieurs unités lexicales il faut[9] l'écrire entre accolades.

## 4.2 Macros pour un gobage ou dégobage inconditionnel

Le paquetage fournit également deux macros de plus bas niveau `\makegobbler` et `\makeungobbler` qu'en général l'utilisateur $\lambda$ n'a pas besoin d'utiliser directement, et qui servent respectivement au gobage et au dégobage inconditionnel d'un bout de code LaTeX.

Je décris ci-après ci-après ces deux macros parce qu'elles constituent le mécanisme sous-jacent du paquetage, c'est à dire que sous le capot, les macros pour le gobage et/ou dégobage conditionnel, `\MGBkeep`, `\MGBdrop`, `\MGBkeepelse`, et

7. ⟨*condition*⟩ correspond à `{`⟨*test*⟩`}` dans le manuel de `ifthen`.
8. Toto a un cousin anglais complètement foo.
9. Ceci n'a rien à voir avec le paquetage `makeglobber`, c'est une règle générale de TeX, lorsqu'un argument de macro consiste en une séquence de plusieurs unités lexicales on le met entre accolades pour le passer d'un seul tenant à la macro. Lorsque il y a une seule unité lexicale, alors les accolades sont optionnelles, on peut passer `{\iftoto}` aussi bien que `\iftoto`.

\MGBdropelse, finissent par se développer, selon la ⟨*condition*⟩ qui leur est passée, en un \makegobbler ou un \makeungobbler

Le nom du paquetage provient d'ailleurs de ce que le délimiteur \⟨*délimiteur*⟩ peut être utilisé avec \makegobbler pour supprimer tout ce qui suit jusqu'au prochain \⟨*délimiteur*⟩, comme ci-dessous, où pour l'exemple \⟨*délimiteur*⟩ est nommé \LaMacroQuiGobeTout :

```
\makegobbler\LaMacroQuiGobeTout
Bla Bla bla, tout le texte ici est supprimé.
\LaMacroQuiGobeTout
```

On appelle cette suppression *gobage*, et on dit que \LaMacroQuiGobeTout est un gobeur, la macro \makegobbler sert donc à fabriquer un gobeur du nom de \LaMacroQuiGobeTout qui va tout gober jusqu'à rencontrer une marque de fin également nommé \LaMacroQuiGobeTout, sauf que du point de vue catcodique la première occurrence de \LaMacroQuiGobeTout est une séquence de contrôle, alors que la seconde est une string, c'est à dire une suite de caractères de catcode other.

Si on avait utilisé \makeungobbler à la place de \makegobbler alors le texte aurait été inséré au lieu d'être supprimé :

```
\makeungobbler\LaMacroQuiGobeTout
Bla Bla bla, tout le texte ici est inséré.
\LaMacroQuiGobeTout
```

Et donc dans l'exemple ci-dessus on parle de *dégobage* et on dit que \LaMacroQuiGobeTout a été définie comme un dégobeur, c'est à dire qu'elle conserve tout ce qui suit jusqu'à la prochaine occurrence de \LaMacroQuiGobeTout qui sert de marque de fin, et que cette marque de fin est supprimée.

Il est à noter que la définition de \⟨*délimiteur*⟩ comme une macro de gobage/dégobage est transitoire. Une fois la marque de fin consommée, \⟨*délimiteur*⟩ reprend sa valeur initiale si elle en avait une, ou sinon redevient indéfini.

# 5 Le code

## 5.1 Options du paquetage

Les déclarations habituelles pour que LaTeX connaisse le paquetage.

```
\NeedsTeXFormat{LaTeX2e}[2020/10/01]
\ProvidesPackage{makegobbler}
  [2025-11-29 v0.3 %
    Gobbling macros maker]
```

\MGB@process@ifthen@option Une option juste pour dire de charger le paquetage ifthen qui permet d'avoir des conditions à la LaTeX. La macro \MGB@process@ifthen@option sert juste à faire le \RequirePackage{ifthen} après le \ProcessOptions, c'est à dire qu'on la développe à cet endroit là. Elle s'autodétruit assitôt sa tâche accomplie, vu qu'on n'en a plus besoin par la suite.

```
\DeclareOption{ifthen}{\def\MGB@process@ifthen@option
  {\RequirePackage{ifthen}\let\MGB@process@ifthen@option\@undefined}}
```

Si l'option `ifthen` n'est pas passée, `\MGB@process@ifthen@option` ne fait rien mis à part s'autodétruire.

```
\def\MGB@process@ifthen@option{\let\MGB@process@ifthen@option\@undefined}
```

N'utilisez pas l'option `tense`, sauf si vous avez lu et compris le code qui l'utilise, et si vous recherchez un substitut à la caféine.

```
\newif\ifMGB@tense
\DeclareOption{tense}{\MGB@tensetrue}

\ProcessOptions*
\MGB@process@ifthen@option
```

## 5.2 Interface utilisateur

### 5.2.1 Macros de gobage/dégobage conditionnel

`\MGBkeep` `\MGBdrop` `\MGBkeepelse` `\MGBdropelse`

Définition des macros d'interface utilisateur, les quatre définitions ont sensiblement la même gueule parce qu'en premier lieu on ne fait que sélectionner si on passe une condition à la TEX en étoilant la macro, ou bien à la LATEX en ne l'étoilant pas.

```
\def\MGBkeep{\MGB@stdhooks\@ifstar{\MGBkeep@tex}{\MGBkeep@latex}}
\def\MGBdrop{\MGB@stdhooks\@ifstar{\MGBdrop@tex}{\MGBdrop@latex}}
\def\MGBkeepelse{\MGB@elsehooks\@ifstar{\MGBkeep@tex}{\MGBkeep@latex}}
\def\MGBdropelse{\MGB@elsehooks\@ifstar{\MGBdrop@tex}{\MGBdrop@latex}}
```

### 5.2.2 Macros de gobage/dégobage inconditionnel

Les quatre macros de gobage/dégobage conditionnel sous le capot finissent tôt ou tard par développer l'une des deux macros de plus bas niveau `\MGB@g` dans le cas d'un gobage, ou `\MGB@u` dans le cas d'un dégobage. Ces deux dernières sont également accessible depuis l'interface utilisateur, via respetivement les macros `\makeglobber` et `\makeunglobber`, en effet `\makeglobber` pourrait être utile par exemple en DocTEX pour gober la section pilote du document `.dtx`.

`\makegobbler` `\makeungobbler`

Les macros `\makegobbler` et `\makeungobbler` délèguent le traitement proprement dit de gobage/dégobage à respectivement `\MGB@g` et `\MGB@u` avec le préliminaire de développer en premier lieu `\MGB@stdhooks`, ce dont on verra l'utilité en § 5.3 :

```
\def\makegobbler{\MGB@stdhooks\MGB@g}%
\def\makeungobbler{\MGB@stdhooks\MGB@u}%
```

`\MGB@u` La macro `\MGB@u` a juste pour effet qu'elle gobe le délimiteur, et le définit de sorte qu'il se redéfinisse à sa valeur initiale. Donc ce qui suit entre les deux délimiteurs n'est pas gobé.

```
\def\MGB@u#1{\MGB@save@mark#1\def#1{\MGB@ungobblerhook#1}}%
```

`\MGB@g` La macro `\MGB@g` prend en argument une séquence de contrôle \⟨*delimiteur*⟩, et gobe tout ce qui suit jusqu'à rencontrer la string \⟨*delimiteur*⟩ :

```
\def\MGB@g#1{%
  \MGB@save@mark#1%
  \begingroup
  \expandafter\MGB@\expandafter
  {\string#1}#1\MGB@set@catcoderegime#1}%
```

## 5.3 Gestion des crochets de fin de traitement

Notez que les deux macros `\MGBkeepelse` et `\MGBdropelse` ne se distinguent respectivement de `\MGBkeep` et `\MGBdrop` que par l'insertion de la macro `\MGB@elsehooks` au lieu de `\MGB@stdhooks`. En fait le cœur du réacteur c'est les macros `\MGB@g` et `\MGB@u`. Chacune des ces macros développe en fin de traitement respectivement une macro crochet `\MGB@gobblerhook`, resp. `\MGB@ungobblerhook`, dont la valeur standarde `\MGB@stdhook` consiste à finir le traitement.

`\MGB@stdhooks` La macro `\MGB@stdhooks` règle les définitions de ces deux macros crochet à leur valeur standarde, après avoir empilé la valeur courante :

```
\def\MGB@stdhooks{%
```

Donc en premier lieu, avant de faire ce réglage, elle empile la valeur courante des macros crochets, ceci permettant de cascader plusieurs niveaux de gobage/dégobage avec leur $\backslash\langle$*délimiteur*$\rangle$ respectifs :

```
  \MGB@pushhooks
```

puis elle fait le réglage proprement dit aux valeurs standardes des crochets en développant la macro`\MGB@standardizehooks` :

```
  \MGB@standardizehooks
  }
```

`\MGB@standardizehooks` La macro `\MGB@standardizehooks` règle les définitions des deux macros crochet à leur valeur standarde, c'est à dire celle qui termine le traitement, ou en d'autres termes celle qui correspond au fait que le $\backslash\langle$*délimiteur*$\rangle$ est le dernier :

```
\def\MGB@standardizehooks{%
  \let\MGB@gobblerhook\MGB@stdhook
  \let\MGB@ungobblerhook\MGB@stdhook
}
```

`\MGB@elsehooks` La macro `\MGB@elsehooks` ne fait que changer les définitions de ces macros crochet pour qu'un dégobage soit fait à la suite d'un gobage, ou resp. un gobage à la suite d'un dégobage, ceci permettant de traiter la branche `else`.

```
\def\MGB@elsehooks{%
```

Comme avec `\MGB@stdhooks`, la première chose qu'on fait c'est d'empiler la valeur courante des macros crochet, de sorte à pouvoir les dépiler en fin de gobage.

```
  \MGB@pushhooks
```

Donc on règle le crochet de gobage...

```
  \def\MGB@gobblerhook##1{%
```

pour tout d'abord écraser les valeurs courantes des crochets de gobage/dégobage par ls valeur standardes, sans pour autant changer l'état de la pile...

```
    \MGB@standardizehooks
```

et ensuite faire un dégobage :

```
    \MGB@u##1}%
```

et pour le crochet de dégobage c'est l'inverse...

```
  \def\MGB@ungobblerhook##1{%
```

comme précédement on règle les macros crochet aux valeurs standardes sans changer l'état de la pile...

```
    \MGB@standardizehooks
```

mais ensuite on enchaîne sur un gobage au lieu d'un dégobage :

```
    \MGB@g##1}%
}
```

`\MGB@stdhook` La macro `\MGB@stdhook` est la valeur par défaut des macros crochet, elle finit le traitement de gobage/dégobage : c'est à dire elle restaure la macro `\`⟨*délimiteur*⟩ à sa valeur initiale [10] :

```
\def\MGB@stdhook#1{%
  \MGB@pophooks
  {\def\@tempa##1{\let#1##1\let##1\@undefined}%
    \expandafter\expandafter\expandafter}\expandafter
  \@tempa\csname MGB@save\string#1\endcsname
}%
```

`\MGB@save@mark` La restauration se fait à partir d'une sauvegarde faite dans la variable [11] `\MGB@save\`⟨*délimiteur*⟩. La macro `\MGB@save@mark` sert à effectuer la sauvegarde en début de traitement :

```
\def\MGB@save@mark#1{%
  \expandafter\let\csname MGB@save\string#1\endcsname#1}%
```

## 5.4 Gestion de la pile de macros crochet

`\MGB@hook@sd` La macro `\MGB@hook@sd` est un compteur qui donne la profondeur dans la pile de macros crochet (`sd` = «**s**tack **d**epth») :

```
\newcount\MGB@hook@sd
```

`\MGB@pushhooks` La macro `\MGB@pushhooks` sert à empiler la valeur courante des macros crochet :

```
\def\MGB@pushhooks{%
  \expandafter\let\csname MGB@ghookstack\the\MGB@hook@sd\endcsname\MGB@gobblerhook
  \expandafter\let\csname MGB@uhookstack\the\MGB@hook@sd\endcsname\MGB@ungobblerhook
  \advance\MGB@hook@sd\@ne
}
```

`\MGB@pophooks` La macro `\MGB@pophooks` sert à dépiler les valeurs des macros crochet de sorte à restituer les valeurs avant le dernier empilement :

```
\def\MGB@pophooks{%
  \advance\MGB@hook@sd\m@ne
  \expandafter\let\expandafter\MGB@gobblerhook
    \csname MGB@ghookstack\the\MGB@hook@sd\endcsname
  \expandafter\let\expandafter\MGB@ungobblerhook
    \csname MGB@uhookstack\the\MGB@hook@sd\endcsname
}
```

Touche finale de la gestion de la pile, les macros crochet de gobage/dégobage sont initialisées pour produire une erreur indiquant un épuisement de la pile [12]. En effet, si on ne fait pas cela, et que par malheur survient une situation où se produit un dépilement de plus que d'empilements, alors on aura un message d'erreur abscons indiquant soit que `\MGB@gobblerhook` n'est pas définie, soit que c'est `\MGB@ungobblerhook` qui n'est pas définie. Avec les initialisations suivantes, dans une telle situation au lieu de cela on a un message d'erreur beaucoup plus explicite :

```
\def\MGB@gobblerhook#1{%
  \PackageError{makegobbler}{\'Epuisement pile macros crochet}{%
    Une erreur a produit un d\'epilage de plus que d'empilage.%
  }
}
\let\MGB@ungobblerhook\MGB@gobblerhook
```

10. Sa valeur avant gobage (`\MGB@g`) ou dégobage (`\MGB@u`).
11. La deuxième contr'oblique est une lettre.
12. «stack underflow» en langue anglaise

## 5.5 Gobage/dégobage dans le cas d'une condition à la TEX

\MGBkeep@tex
\MGBdrop@tex

On commence par le gros morceau, le cas des conditions à la TEX. En fait on veut grosso modo que `\MGBkeep*\ifcondition` se développe en :

```
\ifcondition
⟨couic⟩
  \expandafter\MGB@u
\else
  \expandafter\MGB@g
\fi
```

Dans le cas de `\MGBdrop` c'est juste l'ordre de `\MGB@u` et `\MGB@g` qui est inversé. Comme c'est un peu coton de mettre en clair un `\else` ou un `\fi` dans un corps de macro sans qu'il n'y ait `\ifcondition` également en clair, on va passer le `\else` et le `\fi` via des registres toks pour qu'ils ne soient pas en clair. Pour faire ça on se place au sein d'un groupe, et le boulot sera fait via une macro `\@tempa` développée en after-group.

```
\begingroup
```

Au fait, `\ifcondition` ça peut être n'importe quelle condition à la TEX, par exemple `\ifnum`⟨*a*⟩=⟨*b*⟩␣. Du coup pour éviter qu'au cas où l'utilisateur oublierait l'espace[13] juste après le ⟨*b*⟩ et où donc la branche vraie serait intempestivement développée entrainant une erreur de type `\else` sans `\if`, on insère en général un `\relax` après le `\ifcondition`, sauf pour les utilisateurs qui aiment vivre dangereusement et ont passé l'option `tense` au paquetage. Donc en général `\toks0` contient `\relax`, sauf si l'option `tense` et passée, auquel cas il est vide.

```
\edef\@tempa{\toks0{\ifMGB@tense\else\relax\fi}}\@tempa
```

`\toks1` contient `\else`, et

```
\toks1\expandafter{\csname else\endcsname}%
```

`\toks2` contient `\fi`.

```
\toks2\expandafter{\csname fi\endcsname}%
```

Et maintenant on définit `\MGBkeep@tex`, et `\MGBdrop@tex` dans une macro `\@tempa` développée en after-group.

```
\edef\@tempa{%
  \def\noexpand\MGBkeep@tex##1{##1\the\toks0
    \noexpand\expandafter\noexpand\MGB@u
    \the\toks1
    \noexpand\expandafter\noexpand\MGB@g
    \the\toks2
  }%
  \def\noexpand\MGBdrop@tex##1{##1\the\toks0
    \noexpand\expandafter\noexpand\MGB@g
    \the\toks1
    \noexpand\expandafter\noexpand\MGB@u
    \the\toks2
  }%
}%
\expandafter\endgroup\@tempa
```

## 5.6 Gobage/dégobage dans le cas d'une condition à la LATEX

\MGBkeep@latex
\MGBdrop@latex

Le cas des conditions à la LATEX est beaucoup plus simple à traiter. En fait on

13. Cet espace fait partie du nombre et ne génère pas une espace dans le texte.

veut grosso modo que `\MGBkeep{`⟨*condition*⟩`}` se développe en :

```
\ifthenelse{⟨condition⟩}{%
  \MGB@u}{%
  \MGB@g}
```

Pour développer `\MGB@u` si ⟨*condition*⟩ est vraie, ou `\MGB@g` sinon. Et pour `\MGBdrop` c'est l'inverse, ce qu'on obtient en intervertissant `\MGB@u` et `\MGB@g`.

```
\def\MGBkeep@latex#1{\ifthenelse{#1}{\MGB@u}{\MGB@g}}
\def\MGBdrop@latex#1{\ifthenelse{#1}{\MGB@g}{\MGB@u}}
```

## 5.7 Macros auxliaires de `\MGB@g`

`\MGB@set@catcoderegime`

La macro `\MGB@set@catcoderegime` sert à changer le régime de catcode pour que dans le cas du gobage, tout ce qui est gobé soit pris verbatim.

```
\def\MGB@set@catcoderegime{%
  \catcode35=12
  \catcode123=12
  \catcode125=12
  \catcode37=12
  \catcode92=12
}
```

`\MGB@otherize` La macro `\MGB@otherize` change le catcode de chacun des caractères du délimiteur en other. C'est pour que le délimiteur soit bien reconnu comme marque de fin.

```
\def\MGB@otherize#1{%
  \def\@tempa{#1}%
  \ifx\@tempa\@nnil\else
    \catcode`#1=12
    \expandafter\MGB@otherize
  \fi
}
```

`\MGB@` La macro `\MGB@` est l'auxiliaire de `\MGB@g`, elle prend deux arguments :

- `#1` Le délimiteur en tant que string
- `#2` Le délimiteur en tant que séquence de contrôle

```
\def\MGB@#1#2{\MGB@otherize#1\@nil\long\def#2##1#1{\endgroup
    \MGB@gobblerhook#2}}%
```

E finita la comedia !

```
\endinput
```